(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DIGITAL SIGNAL PROCESSING SOFTWARE
PACKAGES FOR IBM-PC AND
IBM-PC WITH DSP-16

by

Gregory James Pitman

December 1989

Thesis Advisor: Murali Tummala

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) DIGITAL SIGNAL PROCESSING SOFTWARE PACKAGES FOR IBM-PC AND IBM-PC WITH DSP-16

12 PERSONAL AUTHOR(S)
PITMAN, Gregory James

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1989 December	59

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Digital Signal Processing; DSP; Ariel DSP-16

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis provides two versions of a digital signal processing (DSP) program that can be used by an individual interactively. The intended user for this program is a student taking a digital speed signal processing class at the Naval Postgraduate School. Nevertheless, the implemented algorithms can be used to analyze any slowly time-varying signal. One version uses a MS-DOS computer (or IBM-compatible) for both data handling and for the DSP algorithms. The other version uses the MS-DOS computer for data handling and uses an Ariel DPS-16 digital signal processing board for the DSP algorithms. All algorithms are implemented in a non real-time mode. The program uses the same user-interface software for both versions. The user-interface software package provides a user-friendly input and output environment.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL MURALI TUMMALA	22b TELEPHONE (Include Area Code) 408-646-2645 22c OFFICE SYMBOL 62Tu

DD Form 1473, JUN 86 Previous editions are obsolete SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603 UNCLASSIFIED

Approved for public release; distribution is unlimited

Digital Signal Processing Software Packages for IBM-PC
and IBM-PC with DSP-16

by

Gregory James Pitman
Lieutenant Commander, United States Navy
B.A., Culver-Stockton College, 1976

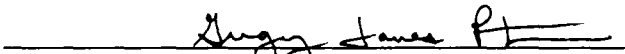
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

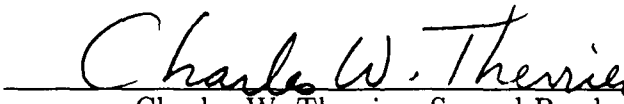
from the

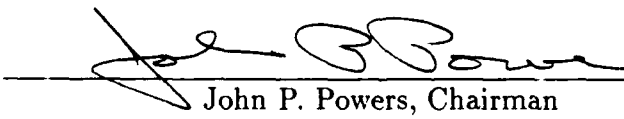
NAVAL POSTGRADUATE SCHOOL

December 1989

Author: 
Gregory James Pitman

Approved by: 
Murali Tummala, Thesis Advisor


Charles W. Therrien, Second Reader


John P. Powers, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

This thesis provides two versions of a digital signal processing (DSP) program that can be used by an individual interactively. The intended user for this program is a student taking a digital speech signal processing class at the Naval Postgraduate School. Nevertheless, the implemented algorithms can be used to analyze any slowly time-varying signal. One version uses a MS-DOS computer (or IBM-compatible) for both data handling and for the DSP algorithms. The other version uses the MS-DOS computer for data handling and uses an Ariel DSP-16 digital signal processing board for the DSP algorithms. All algorithms are implemented in a non real-time mode. The program uses the same user-interface software for both versions. The user-interface software package provides a user-friendly input and output environment.

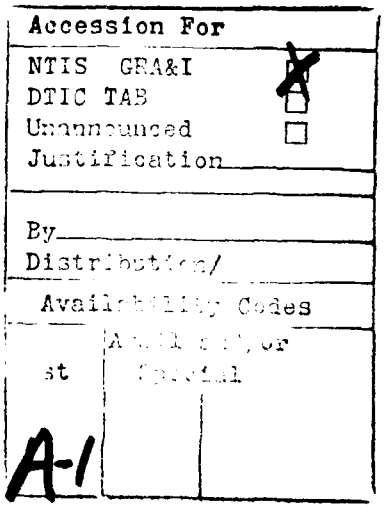


TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

Hercules is a registered trademark of Hercules Computer Technology.

IBM is a registered trademark of International Business Machines Corporation.

Microsoft, MS-DOS, and QuickC are registered trademarks of Microsoft Corporation.

Olivetti is a registered trademark of Ing C. Olivetti.

I. INTRODUCTION

There exist many computer environments that allow a user to perform digital signal processing (DSP). Some of these environments require the individual to program every step of the desired algorithm. These are typified by high-level programming languages like FORTRAN, Pascal, or C. Other environments provide powerful built-in functions that can be used by the individual to program the algorithm. This type of environment is typified by so-called very high level languages such as APL and Matlab.

One problem that arises from either of these environments is that they all require programming. Additionally, no common basis for input of the signal nor output of the results is provided without additional programming. This programming can be quite time-intensive. In a classroom environment where the purpose is to understand the theory of the various DSP algorithms and then demonstrate them in the laboratory, this programming by each individual is not productive.

The objective of this thesis is to provide a DSP program that can be used by a user and not require programming by that user. The typical user is expected to be a student who is learning the theory behind the implemented algorithms. The proposed software is meant for classroom use in digital signal processing courses at this institution, "EC4410-Speech Signal Processing" in particular. Additionally, the program should be able to utilize an Ariel DSP-16 digital signal processing board in a pseudo-coprocessor role. These objectives can be subdivided into the following goals:

- Provide a common user-friendly interface that simplifies input and output.

- Develop a package of algorithms for the host machine and demonstrate one for the DSP-16 in a non-real time mode that is interfaced from the host machine.

In order to meet the needs of EC4410, the algorithms in this DSP program deal with analysis of speech signals. However, the program is suitable for analyzing any slowly time-varying signals.

The remaining part of this thesis addresses these objectives and goals and shows how they were met. A brief outline of the chapters follows.

Chapter II provides background material for understanding the software that is developed in later chapters. It begins by providing a brief review of the DSP algorithms implemented. Next it reviews the hardware requirements of the host computer and its peripherals. Finally, the chapter covers the layout of the DSP-16 board and the functional layout of the Texas Instruments' TMS320C25.

Chapter III presents the design and development of the user-interface software. Included in this chapter is a discussion of the software engineering aspects of the design and the general layout of the input and output functions. The last section discusses auxiliary functions that do not fall into the other categories.

Chapter IV deals with the software modules that actually perform the DSP algorithms. The first section discusses the software that allows the PC to perform the algorithms. The last section discusses the host interface software and DSP-16 software. This last discussion deals with the programming and debugging considerations for the host and the DSP-16.

Chapter V contains the overall conclusions of this thesis. This chapter includes a discussion of software and hardware limitations discovered in the course of this research. Additionally, further possible developments are reviewed.

II. BACKGROUND

This chapter provides background material covering the digital signal processing algorithms implemented, the computer system required by the program, a description of the DSP-16, and a discussion of the TMS320C25 processor used on the DSP-16 system.

A. DIGITAL SIGNAL PROCESSING ALGORITHMS

In this section we briefly describe the digital signal processing algorithms that the program developed here performs. The intention is only to summarize these algorithms *briefly* in the following sections; the details may be found in [Refs. 1, 2, 3]. All of these algorithms are assumed to be performed in a non real-time environment.

1. Fast Fourier Transforms

The Fast Fourier Transform (FFT) algorithm implements the discrete Fourier transform (DFT), given by [Ref. 1, Chapter 8]

$$X(k) = \sum_{n=0}^{N-1} x(n)[W_N]^{nk}, k = 0, 1, 2, \ldots, N-1 \tag{2.1}$$

where $W_N = e^{-j(2\pi/N)}$ and N is a power of 2. The W_N term is referred to as the twiddle factor. The FFT algorithm can more easily be viewed as an expanding butterfly where the representation for a butterfly is given in Figure 2.1. However, in order to use this method, the input or the output must have a bit-reversed sorting performed. This is accomplished by indexing the values from 0 to $N-1$, then the binary value of the index is reversed. The values stored in the original position and the bit reversed position are exchanged. This implementation, which is called decimation in time, will assume that the input values are bit reversed. Figure 2.2 shows a formulation for the

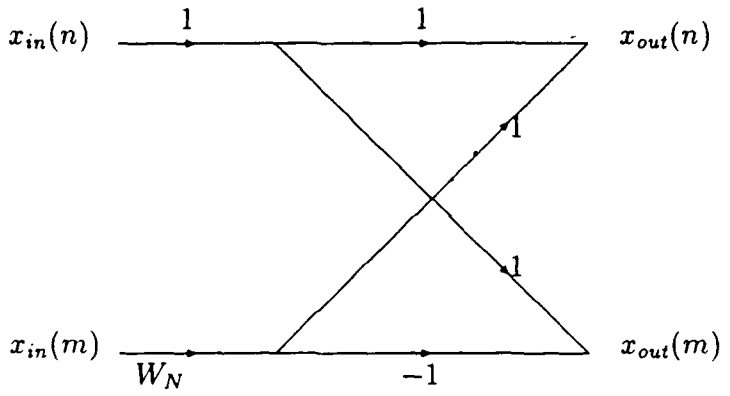


Figure 2.1: Butterfly representation

FFT calculation. The outer loop determines that there will be $N_s = \log_2 N$ stages performed. The middle loop determines for each stage how many 2^{N_s}-point FFTs must be performed. The inner loop performs the 2^{N_s-1} butterflies. The function Butterfly has three arguments. The first argument corresponds to $x_{in}(n)$, the second to $x_{in}(m)$, and the third to W_N as given in the butterfly representation (Fig. 2.1).

```
for s = 1 to log2 N
    q = 2^s
    m = 2^(s-1)
    for k = 0 to (N/q - 1)
        for n = 0 to (m - 1)
            Butterfly( x(k*q + n), x(k*q + n + m), W(n,q) )
        end for
    end for
end for
```

Figure 2.2: Multi-Stage FFT Calculation

2. Time Domain Algorithms

The time domain algorithms are straightforward to develop. Additionally, the window function $w(n)$ in the following equations will be assumed to be a rectangular window with a magnitude of one and a size of N points.

The first of the time domain functions is the short-time energy, given by

$$E_n = \sum_{m=-\infty}^{\infty} [x(m)w(n-m)]^2. \tag{2.2}$$

By applying the rectangular window to Equation 2.2, it can be simplified to

$$E_n = \sum_{m=n}^{m-N+1} x(m)^2. \tag{2.3}$$

The second algorithm is the short-time magnitude function

$$M_n = \sum_{m=-\infty}^{\infty} \mid x(m) \mid w(n-m). \tag{2.4}$$

The rectangular window can be applied to Equation 2.4, to give

$$M_n = \sum_{m=n}^{m-N+1} \mid x(m) \mid . \tag{2.5}$$

The third algorithm is the short-time zero-crossing function

$$Z_n = \sum_{m=-\infty}^{\infty} \frac{\mid sgn[x(m)] - sgn[x(m-1)] \mid}{2} w(n-m) \tag{2.6}$$

where

$$sgn[x(n)] = \begin{cases} 1 & \text{if } x(n) \geq 0 \\ -1 & \text{if } x(n) < 0. \end{cases} \tag{2.7}$$

By applying the rectangular window to Equation 2.6, the algorithm can be simplified to

$$Z_n = \sum_{m=n}^{m-N+1} \frac{\mid sgn[x(m)] - sgn[x(m-1)] \mid}{2}. \tag{2.8}$$

The fourth formula implemented is the short-time autocorrelation given by

$$R_n(k) = \sum_{m=-\infty}^{\infty} x(n+m)x(n+m+k)[w(m)w(k+m)] \tag{2.9}$$

where k is lag of the autocorrelation and it indicates the offset between the two values being correlated. When the rectangular window is applied to Equation 2.9, we have the final form

$$R_n(k) = \sum_{m=0}^{N-k-1} x(n+m)x(n+m+k). \tag{2.10}$$

The fifth and last algorithm is short-time 3-level autocorrelation

$$R_n(k) = \sum_{m=0}^{N-k-1} y(n+m)y(n+m+k). \tag{2.11}$$

In addition to the lag variable k, this algorithm has a variable labeled L_c, the clip level. If the sample value does not have a magnitude greater than the clip level, the sample value will be set to zero. The clipping is performed as follows

$$y(n) = \begin{cases} 1 & \text{if } x(n) > L_c \\ -1 & \text{if } x(n) < -L_c \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

With Equation 2.12, Equation 2.11 can be reduced one step further to eliminate all multiplications. The result is

$$y(n+m)y(n+m+k) = \begin{cases} 0 & \text{if } y(n+m) = 0 \text{ or } y(n+m+k) = 0 \\ 1 & \text{if } y(n+m) = y(n+m+k) \\ -1 & \text{otherwise.} \end{cases} \tag{2.13}$$

3. Linear Predictive Coding Algorithms

This section presents three types of linear predictive coding (LPC) algorithms. Here we summarize these algorithms to gain an insight into implementing them in software. The details are available in the literature [Refs. 2, 3].

The first of these algorithms is Durbin's method and is also referred to as the Levinson method [Ref. 3, pages 202-210]. An initial calculation that must be performed prior to applying this method is the computation of the autocorrelation of the input sequence where the lag equals the desired order of the predictor. The results of the autocorrelation will be available as a sequence $R(0) \ldots R(p)$ where p is the desired order of the filter. The procedure is as follows:

1. Initialize: $A_0(z) = 1,\ E_0 = R(0)$.

2. For $s = 1$: p at stage s, the filter $A_s(z)$ and error E_s are available.

3. Compute the "reflection coefficient" γ_{s+1} using

$$\gamma_{s+1} = \frac{\Delta_s}{E_s} \tag{2.14}$$

where $\Delta_s = \sum_{i=0}^{p} a_{s,i} R(s+1-i)$.

4. Determine the new polynomial $A_{s+1}(z)$ using

$$\begin{bmatrix} 1 \\ a_{s+1,1} \\ a_{s+1,2} \\ \vdots \\ a_{s+1,s} \\ a_{s+1,s+1} \end{bmatrix} = \begin{bmatrix} 1 \\ a_{s,1} \\ a_{s,2} \\ \vdots \\ a_{s,s} \\ 0 \end{bmatrix} - \gamma_{s+1} \begin{bmatrix} 0 \\ a_{s,s} \\ a_{s,s-1} \\ \vdots \\ a_{s,1} \\ 1 \end{bmatrix}. \tag{2.15}$$

5. Update the mean-square prediction error E_{s+1} using

$$E_{s+1} = (1 - \gamma_{s+1}^2) E_s. \tag{2.16}$$

6. If $s = p$, then stop. Else s = s + 1 and loop to step 3.

The next algorithm is Burg's method [Ref. 3, pages 259-261]. This algorithm uses the input data sequence $\{x_0, x1, \ldots, x_{N-1}\}$ without having to compute the autocorrelation function. The variable p is the desired final order of the predictor. The procedure is as follows:

1. Initialize: $A_0(z) = 1$, $e_0^+(n) = e_0^-(n) = x_n$, for $0 \leq n \leq N-1$ and

$$E_0 = \frac{1}{N} \sum_{n=0}^{N-1} x_n^2. \tag{2.17}$$

2. For $s = 1 : p$ at stage s, the filter $A_s(z)$, error E_s are available. Set $s = s + 1$.

3. Compute the "reflection coefficient" γ_s using

$$\gamma_s = \frac{2\sum_{n=s}^{N-1} e_{s-1}^{+}(n) e_{s-1}^{-}(n-1)}{\sum_{n=s}^{N-1} [e_{s-1}^{+}(n)^2 + e_{s-1}^{-}(n-1)^2} \tag{2.18}$$

where $e_0^+(n)$ and $e_0^-(n)$ are referred to as the forward and backward prediction error, respectively.

4. Determine the new polynomial $A_s(z)$ using

$$\begin{bmatrix} 1 \\ a_{s,1} \\ a_{s,2} \\ \vdots \\ a_{s,s-1} \\ a_{s,s} \end{bmatrix} = \begin{bmatrix} 1 \\ a_{s-1,1} \\ a_{s-1,2} \\ \vdots \\ a_{s-1,s-1} \\ 0 \end{bmatrix} - \gamma_s \begin{bmatrix} 0 \\ a_{s-1,s-1} \\ a_{s-1,s-2} \\ \vdots \\ a_{s-1,1} \\ 1 \end{bmatrix}. \tag{2.19}$$

5. Update the errors $e_s^+(n)$ and $e_s^-(n)$ for $s \leq n \leq N-1$ using

$$e_s^+(n) = e_{s-1}^+(n) - \gamma_s e_{s-1}^-(n-1) \tag{2.20}$$

$$e_s^-(n) = e_{s-1}^-(n-1) - \gamma_s e_{s-1}^+(n). \tag{2.21}$$

6. Update the mean-square prediction error E_s using

$$E_s = (1 - \gamma_s^2) E_{s-1}. \tag{2.22}$$

7. If $s = p$, then stop. Else loop to step 3.

The last of the three algorithms is the covariance method [Ref. 2, pages 407-411]. Consider that the data sequence to be analyzed is $\{x_0, x1, \ldots, x_{N-1}\}$. We first need to compute the p^{th} order correlation matrix R, where p is the order of the predictor. Since R is symmetric, we only need to compute the lower triangular part of R as follows

$$\begin{bmatrix} R_{00} & 0 & 0 & 0 & \ldots & 0 \\ R_{10} & R_{11} & 0 & 0 & \ldots & 0 \\ R_{20} & R_{21} & R_{22} & 0 & \ldots & 0 \\ R_{30} & R_{31} & R_{32} & R_{33} & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ R_{p0} & R_{p1} & R_{p2} & R_{p3} & \ldots & R_{pp} \end{bmatrix} \tag{2.23}$$

where $R_{ij} = \sum_{m=0}^{N-1} x(i+m)x(j+m)$. The complete matrix can be obtained by copying the elements below the main diagonal onto the entries above it. Note that the correlation matrix R obtained above is symmetric but not Toeplitz. The second step in this procedure is to compute the diagonal matrix D and the lower triangular matrix V as follows:

1. Initialize d_{11} to R_{11}, set $j = 1$.

2. Solve for V_{ji} where $1 \leq j \leq i-1$ using

$$V_{ji} = \frac{R_{ij} - \sum_{k=1}^{j-1} V_{ik} d_{kk} V_{jk}}{d_{jk}}. \tag{2.24}$$

3. Set $i = i + 1$, solve for d_{ii} using

$$d_i = R_{ii} - \sum_{k=1}^{i-1} V_{ik}^2 d_{kk}. \tag{2.25}$$

4. If $i > p$ stop, else return to step 2.

The above step is referred to as Cholesky decomposition [Ref. 2, page 407]. In the third step, we compute the terms of the predictor polynomial by the following procedure:

1. Set $X_1 = R_{10}$, $i = 2$.

2. Compute X_i for $2 \leq i \leq p$ using

$$X_i = R_{i0} - \sum_{j=1}^{i-1} V_{ij} X_j. \tag{2.26}$$

3. Set $A_i = X_i / d_{ii}$, $i = n - 1$.

4. Compute A_i for $n-1 \geq i \geq 1$ using

$$A_i = \frac{X_i}{d_{ii}} - \sum_{j=i+1}^{n} V_{ji} A_j. \tag{2.27}$$

The last step in this algorithm is to compute the error terms. This is accomplished as follows (for $0 \leq i \leq p$)

$$E_i = R_{00} - \sum_{k=1}^{i} \frac{X_k^2}{d_{kk}}. \tag{2.28}$$

B. PC AND PERIPHERALS HARDWARE

This section provides a brief description of the required and optional hardware.

- Any computer that is running PC-DOS or MS-DOS version 3.0 or above will be able to use this program. The software should also work on versions of this operating system below 3.0 as well, but has not been tested.

- A math coprocessor of the 80x87 family is supported and preferred, but not required for this software.

- A hard disk is also preferred, but not required.

- The computer must have a video adapter and monitor that can operate in CGA, EGA, VGA, or Hercules graphics mode. Olivetti versions of these modes are also supported. If a Hercules adapter is being used, then the program MSHERC.COM must be executed before using the DSP program [Ref. 4, page 204].

- If a printout of the screen plots is desired, then an Epson-compatible dot matrix printer or HP LaserJet printer is required. Other printer types are not supported. The software can easily be modified to accommodate other types of printers however.

- If the DSP-16 version of the program is desired, then the Ariel DSP-16 board must be installed; this requires one full-size PC expansion slot.

C. DSP-16 BOARD

The DSP-16 board is reviewed in this section. This is followed by a discussion of the Texas Instruments TMS320C25 processor that is installed on the board. The functional layout of the DSP-16 board is shown in Figure 2.3. (For complete details on the board layout, see [Ref. 4, Chapter 6].) This figure shows the main components and busses. In the non real-time mode, the sample rate timer, the analog-to-digital converter(ADC), the digital-to-analog converter(DAC), and the 256K word sample buffer are not required. Though these items are not used by this program, they are initialized by the program. It needs to be pointed out that the TMS320C25 cannot use the sample buffer RAM to hold the program instructions or the data values. The sample buffer is only used to store samples received by the ADC or that will be sent out to the DAC. A description of the remaining items follows.

1. Interface to Host

The interface between the board and the host PC is via a PC expansion slot. The expansion slot pins on the board side are connected to the address lines, the host interrupt, the data lines, and the control register. The address bus of the PC is connected to the address lines on the board. Note that the PC uses 20 address lines while the board uses 14 lines. The PC is a byte-addressable computer whereas the board is a word addressable device. Therefore, the PC can address 64K bytes but the board can only address 16K words. The PC data bus connects to the data lines on the board. This is the standard eight-bit-wide bus on both sides. This apparent problem of addressability and storage is solved by the onboard circuitry. During data transfers to the board, the onboard circuitry will take two bytes from the eight bit

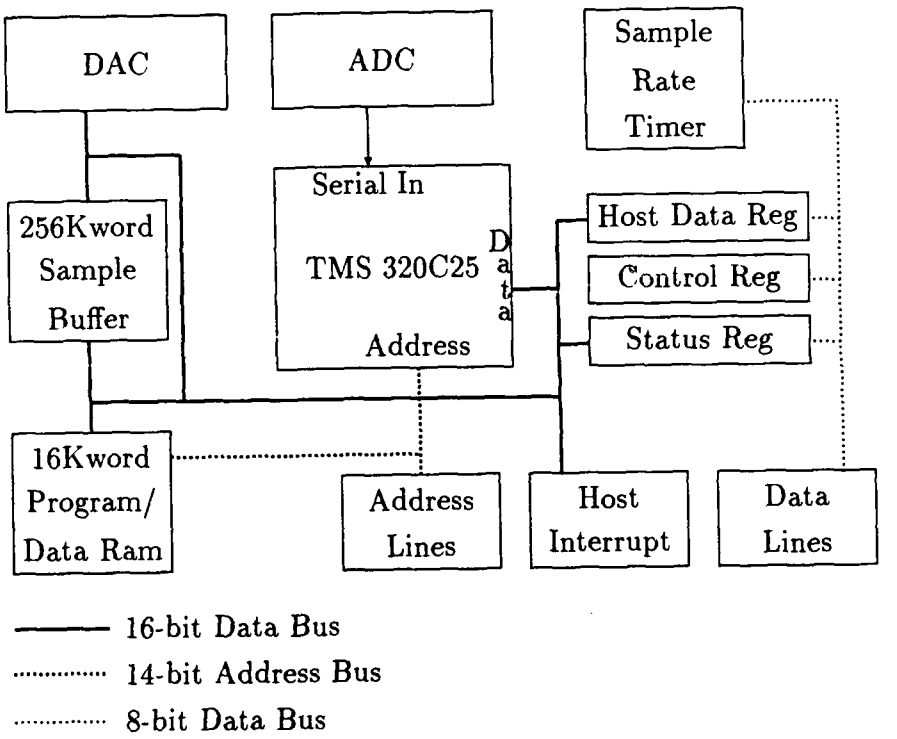


Figure 2.3: Ariel's DSP-16 Board

data bus and transfer them via the onboard 16-bit bus to the onboard storage location specified by the address bus. To achieve the data transfer from the board to the PC the reverse procedure is followed.

In addition to the bus interface, the control register and host interrupt can interact with the PC. The control register can be signaled by the PC via an interface pin to hold the TMS320C25 processor. This feature allows the host PC to transfer to the program/data RAM as if it were native to the PC RAM. The ability to interrupt the host PC from the board is not used by this program. However, this is a feature that may be useful in a real-time mode.

2. Program/Data Ram

The 16K words of Program/Data RAM is a significant aspect of this board. The programming instructions and the data for the algorithm to be executed must be located here. If this is not possible, then the use of program overlays must be considered. In order to use this RAM for data, the GREG register (see section D of this chapter) must be set. By adjusting the setting of the GREG register, the RAM can be divided into varying ratios of program space to data space. Table 2.1 gives the allowable settings. A review of this table shows that the minimum program space is 8192 (decimal) words thereby giving 8192 words of data space. The beginning address for the data space from the TMS320C25's point of view is E000H, but from the host computer's point of view it is still 2000H. Additionally, note that if the program requires 8193 words for storage, then a setting of F0 must be used, thereby losing 4095 words of data storage.

3. Registers

The DSP-16 board has three special use registers. The Status Register contains the host I/O port flags. The TDR flag (bit 1) is set when the board processor writes to the host bus. The HDR (bit 0) is set when the host writes to the board.

The bits are cleared by the appropriate device reading the transferred data. The host data register is used to transfer one word from the board (i.e., two bytes at a time to the PC) without placing the board processor on hold. The reverse can also be done. For either transfer to work however, the processor must poll the status register to check the status of the transfer. The last register is the control register which has already been mentioned.

D. TMS320C25 DSP PROCESSOR

In this section, the Texas Instruments TMS320C25 will be reviewed. This processor is the controller for the DSP-16 board. The functional layout of the processor is shown in Figure 2.4. This layout shows only certain components of the processor that need to be understood for this program and is not meant to be complete. The reference for this section and the processor layout is Reference 6, Chapter 3.

1. Auxiliary Registers

The first cluster of registers on the left of Figure 2.4 make up the auxiliary register group. AR0 through AR7 are the actual auxiliary registers. These registers are 16 bits wide; therefore the largest number that can be represented is 65,536. All numbers in these registers are assumed to be positive values. Below the register stack is the ARAU. This is the auxiliary register arithmetic unit. All arithmetic performed

TABLE 2.1: GREG SETTING (VALUES IN HEX)

GREG Value	Effective Data Location	Actual location in RAM (max range 3FFF)
E0	E000-FFFF	2000
F0	F000-FFFF	3000
F8	F800-FFFF	3800
FC	FC00-FFFF	3C00
FE	FE00-FFFF	3E00
FF	FF00-FFFF	FF00

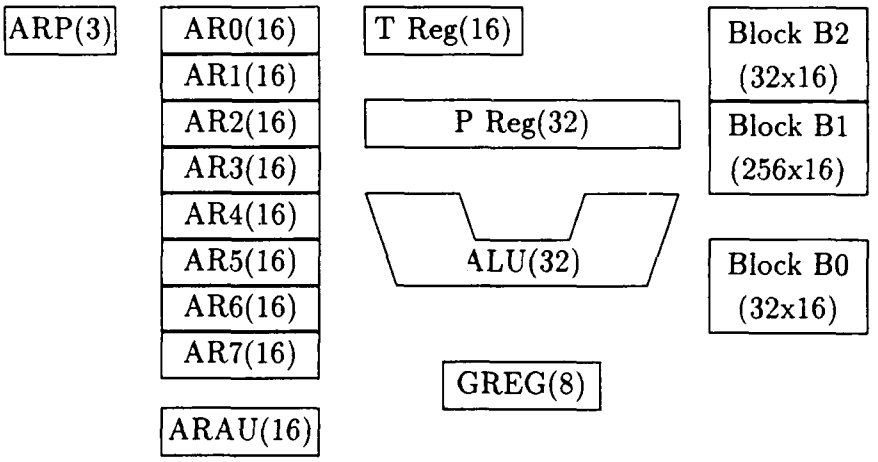


Figure 2.4: TMS320C25 Functional Layout

on an auxiliary register is done by this unit. The last component of this group is the ARP. This three-bit-wide register, known as the auxiliary register pointer, is used to select the current auxiliary register.

The auxiliary registers are useful as pointers to data values or as counters. In the pointer mode, the registers are used in the indirect addressing mode. With indirect addressing, instructions that use the indirect mode can use the value pointed to by the current auxiliary register as one of its operands. This addressing mode is very powerful. With this mode, the pointer can be post-incremented or post-decremented. The size of the change can be one or a value stored in AR0. The arithmetic used in the ARAU can utilize normal carry or reverse carry. The reverse carry allows the bit reversal indexing of FFT values to be performed very rapidly. In the counter mode, the register can be used to the count the number of times an operation is performed.

2. Onboard RAM

On the right of Figure 2.4 there are three blocks of RAM. Block B0 can hold 256 words of either program or data. During program execution this block of RAM can be used either as program RAM or as data RAM but cannot be both types simultaneously. An instruction can be used at runtime to change its status. The other two blocks, B1 and B2, can only be used as data RAM. Block B1 can hold 256 words while block B2 can hold 32. Since all three blocks are onboard the processor, program execution is faster when all data are located in one of these blocks.

3. Registers

The T register, known as the temporary register, is 16 bits wide. The main use of this register is to hold one of the operands of a multiplication operation. The P register, also called the product register, is 32 bits wide. The results of a multiplication operation are stored here. The GREG register, which stands for **g**lobal memory allocation **reg**ister, is eight bits wide. It is used to divide the 16K words of RAM into program and data segments.

With an understanding of the algorithms, the host computer system, and the DSP-16 board that this program utilizes, we are ready to discuss the software development. The first area to look into is the software modules that provide the user interface. These are discussed in the next chapter.

III. USER INTERFACE SOFTWARE

This chapter discusses the software modules that comprise the user interface package. This package include the generic module components, the input/output software, and the auxiliary software. The software packages written for this program use the C language—specifically Microsoft QuickC, Version 2.0. The user manuals that come with this software can be referenced for language constructs and the standard function library [Refs. 5, 6].

A. MODULE COMPONENTS

This program incorporates several aspects of software engineering. The most notable ones are modularity in the user-interface package and commonality among the various modules that make up the package. For example, some of the modules are for the main loop, for the FFT calculations, for the LPC calculations, for the time domain calculations, etc. Within each module, there are functions that perform operations common to functions in the other modules. These similar functions differ in that they use different data sets. This commonality of functions greatly simplifies the development and debugging of software. Other aspects of software engineering can also be noted. A function that is needed by only one particular module is located within that module and is therefore not known to another module. Variables that are needed by only one module are declared only within a function in that module. Some of the common type of functions within the modules are discussed below.

The main type of selection control module is based on a switch statement. A generic example is given in Figure 3.1. The function is called without arguments. The **while** loop is always true, therefore the only exit from this loop is the user selection of

```
int Example( void )
{
int iExCur = 0;
    :   Initialize any variables needed by the function.

while( TRUE )
    {
    iExCur = DisplayMenu( mnuEx, iExCur );

    if( iExCur )
        {
        GetExampleData( ... ) ;
        :   Initialize local variables based on user input.
        }
    /* Branch to menu choice. */
    switch( iExCur )
        {
        case EXQUIT:
            _setvideomode( _DEFAULTMODE );
            return FALSE;
        :   Enumerate all the cases available.
        }
    }
}
```

Figure 3.1: Generic Control Module

"quit". The function DisplayMenu puts a menu on the screen and returns the value of the option the user selected. If the selection was other than quit, then a call is made to the GetExampleData function to get user data. These functions are explained in greater depth in section B of this chapter. The return value from DisplayMenu then selects the corresponding **case** statement which executes the requested action. When the requested action is complete, the program returns to the **while** loop and the loop repeats. Another type of control loop is the kind used for selection of the output display.

The output display functions have a name of the form Plot* where "*" represents a unique name. A generic example of this type is given in Figure 3.2. A comparison of this example with Figure 3.1 shows many similarities. However, there are notable differences between them. The first observation is that the generic output control module has arguments in the function call. A second difference is that there is no need for a GetExampleData function in this module. Once inside a **case** construct, the general layout is to initialize any variables needed, allocate memory, initialize plot labels, call plotting function, and free allocated memory. A benefit of this layout is that any problem associated with an output display is limited to a small area of code.

The last common type of module is the one that controls the calculation of results. These functions have a name of the form Do* where "*" represents a unique name. A generic example of this type is given in Figure 3.3. This function is entered with the information needed for the calculation stored in a C "structure". As information to perform the desired calculation is needed by steps within the function the members of the structure can be accessed to provide the data. The tasks of this function are to allocate needed memory, to calculate the desired algorithm, to call the appropriate output display function, to save the results to the output file, and then to free allocated memory. Functions of the type PlotExample and SaveExampleData

```
int PlotExample( ... )
{
int iCur = 0;
LABEL Label;
    :   Initialize any variables needed by the function.

while( TRUE )
    {
    iCur = DisplayMenu( mnuExPlot, iCur );

    if( iCur )
    /* Branch to menu choice. */
    switch( iCur )
        {
        case EXPLOTQUIT:
            _setvideomode( _DEFAULTMODE );
            return FALSE;
        case PLOTCOEFF:
            :   Initialize any variables needed by the case statement.
                Allocate needed memory dynamically.
            Label.title = pszPlotLabel[EXTITLE];
            Label.xaxis = pszPlotLabel[EXXAXIS];
            Label.yaxis = pszPlotLabel[EXYAXIS];

            PlotOneCont( ... );
            :   Free dynamically allocated memory.

        break;
        :   Enumerate all the remaining cases.
        }
    }
}
```

Figure 3.2: Generic Output Control Module

```
void DoDurbin( struct ...)
{
:   Initialize any variables needed by the case statement.
    Allocate needed memory dynamically.

CalExample( ...);

PlotExample( ...);

SaveExampleData( ...);

:   Free dynamically allocated memory.

return;
}
```

Figure 3.3: Generic Calculation Control Module

are explained in greater depth in Section C. of this chapter. Figure 3.4 gives a flow chart of how these components are jointed together to provide the user interface.

B. INPUT COMPONENTS

The underlying functions for the input components are contained in Fileutil.c and Scrinpt.c. These libraries encapsulate the machine-dependent file system and the video calls, respectively. The concept of isolating machine-dependent functions from the main core of the program is another aspect of software design that is very useful. When these libraries are available, machine-independent C language function calls can be used to manipulate files and display different video options.

In section A of this chapter, the function DisplayMenu was used in order for the user to select one of several options. This function is located in the Scrinpt.c library. The input arguments to this function are an array of structures of type ITEM and an integer value. This structure is defined in the header file Scrinpt.h. An example

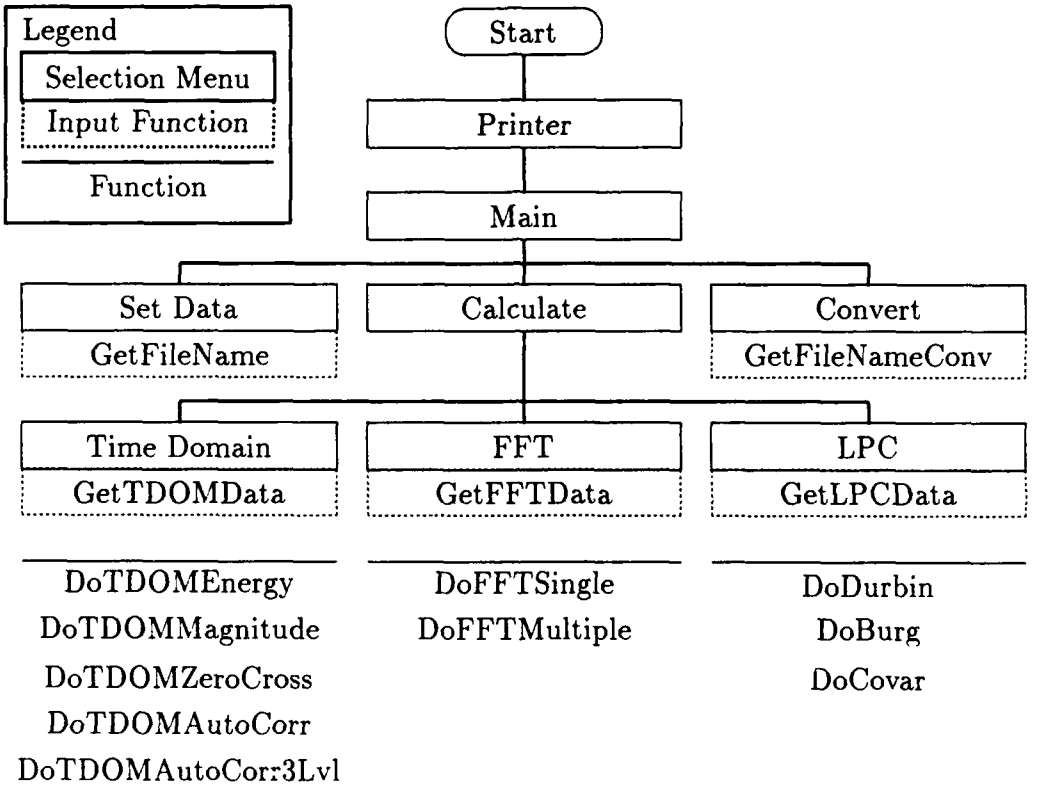


Figure 3.4: User Interface Module Flow Chart

of a structure of type ITEM is given in Figure 3.5. The structure contains two fields for each selection. The first field identifies which character position in the selection is the *hot key.* A hot key is a key or combination of keys that selects a predetermined

```
struct ITEM mnuEx[] =
{                       /* Highlight Char Pos */
0,   "Quit",            /* Q  0 */
0,   "Select 1",        /* S  0 */
3,   "Select 2",        /* 1  2 */
0,   NULL
};
```

Figure 3.5: Example of an Array of ITEM Structure

action. In this case that action is a menu selection and must be unique for each possible menu selection. The second field gives the character string to be displayed for that selection. The integer value controls which selection will be the default. The last entry in the array must be NULL to signify that the last entry in the structure has been reached. In the example, the first selection would be Quit and the hot key is "Q". The desired selection can be made in one of two ways. The first method is to press the desired hot key. The second method is to use the up/down arrow keys to highlight the desired selection and then press the "ENTER" key. At this point the function will return the index of the selection to the function that made the call. Figure 3.6 represents the appearance of the screen if the DisplayMenu function was called with the first argument mnuEx[] (note that [] is the C notation for an array) and the second argument zero. The next function that will be needed for input module is one that accepts the input data.

The purpose of the functions that have the naming convention of Get*Data, where "*" is a unique name, is to accept the user entered values. The general organization of this function is given in Figure 3.7. This type of function has one argument

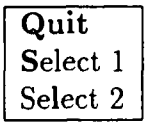


Figure 3.6: Example of a Selection Menu

for each question that is to be asked and can have additional arguments if needed. There are four types of input functions that can be used from within a data input function. They are listed in Table 3.1 together with the kind of data that can be entered and the structure type used by the function. These structure types are defined in Scrinpt.h. All of the input functions display a default value that can be accepted

TABLE 3.1: INPUT FUNCTION TYPES

Function	Entry data Type	Input Structure
InputString	Character string	ISTRING
InputChar	Character	ICHAR
InputInt	Integer	IINT
InputFloat	Float	IFLOAT

by pressing "ENTER" or accept a new value which can be entered as desired. The integer and float function can be defined to check the entered value in one of four ways. The entered value can be checked against a value range, a minimum value, or a maximum value or not checked at all. The general layout of this type of function is as follows:

- Initialize si.ihieght.
- Call the function InitInputForm to setup the entry screen.
- Use PrintAt to output the title of input form.
- Save any values from the input structure that need to be changed for this input. Change default values, if necessary.

```
BOOL GetExampleData( ITITLE *iTitle, IINT *iInt1, ...)
{
int itemp1,itemp2;
long bgColor; /* Screen color, position, and cursor */
struct rccoord rcCursor;
:    Initialize any variables needed by the case statement.

si.ihieght = set equal to number of input lines;

InitInputForm( &bgColor, &rcCursor);

PrintAt( iTitle->row, iTitle->col, iTitle->achTitle, mnuAtrib.fgBorder );

itemp1 = iInt1->iMin;
itemp2 = iInt1->iMax;
iInt1->iMin = 2;
iInt1->iMax = 1000;

InputInt( iInt1 );

iInt2->iMin = itemp1;
iInt2->iMax = itemp2;

:    Add function calls for each input.

CloseInputForm( bgColor, rcCursor);

return FALSE;
}
```

Figure 3.7: Example of a Data Input Function

- Call one of the four input functions.
- Restore default values.
- Repeat last three steps for all input questions.
- Call function CloseInputForm to close out the entry form.

C. OUTPUT COMPONENTS

The underlying functions for the output components are contained in the libraries Cgraph.c and Dspplot.c. The Cgraph.c library encapsulates the machine dependent graphing functions for the video and printers. The Dspplot.c library provides a function that controls the video plotting and the printer plotting, if a printer is attached. The Dspplot.c library contains two plotting functions. One function PlotOneCont does continuous line plotting; another called PlotOneDisc does discrete point plotting. The PlotOneDisc is not used in this program, but is available if needed later.

The function PlotOneCont was initially mentioned in section A of this chapter. This function has five arguments. The five arguments are an array of x-axis points, an array of y-axis points, the number of points in the arrays, a structure containing the label for the plot, and a boolean value to indicate if the data is normalized. The points of the two arrays are (x,y) pairs. After the plot is displayed and if a printer is attached, the user may choose to print the screen to the printer. The user may accept the default response of No or press "Y" to print. If no printer is attached, the response will be "Press any key to continue". Figures 3.8 and 3.9 give examples of the HP LaserJet and Epson output, respectively. The aspect ratio of the output and the screen are different due to the difference between the aspect ratio of a pixel and a dot on the two different printers.

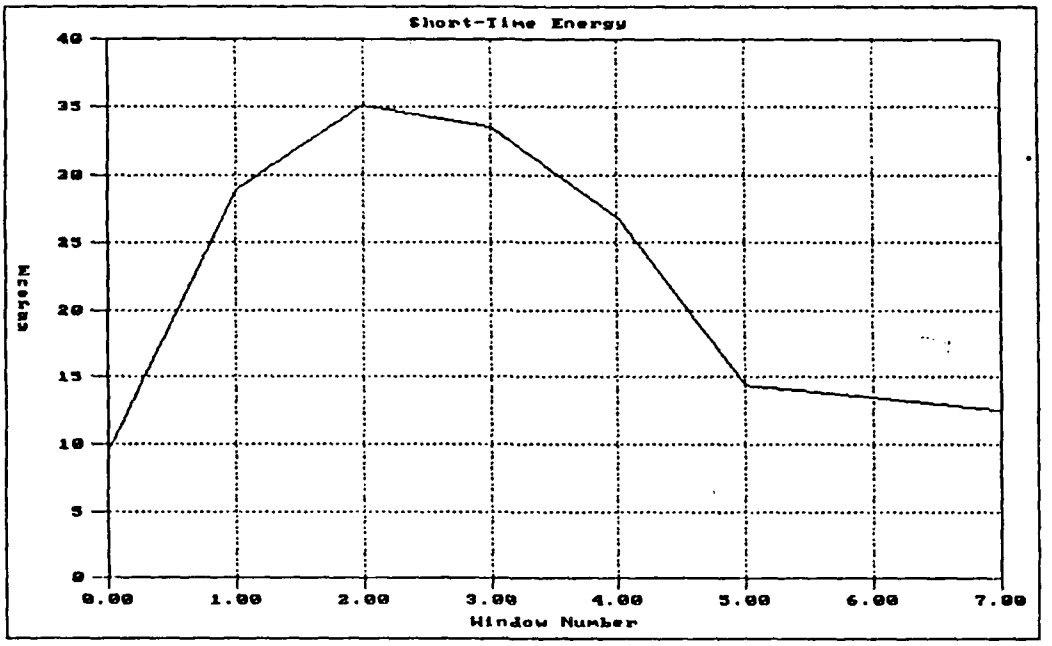


Figure 3.8: HP LaserJet Output with 85% Reduction

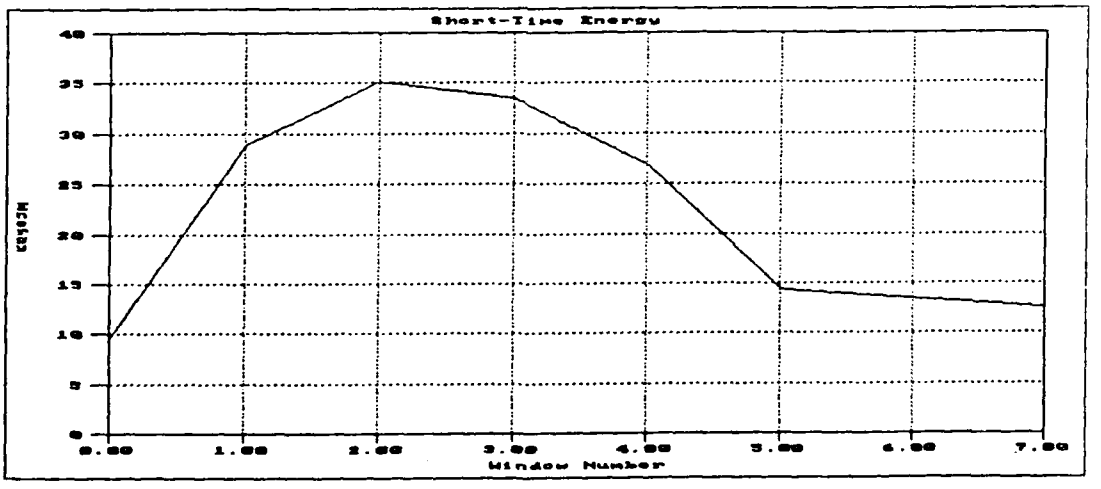


Figure 3.9: Epson Output with 65% Reduction

The development of Cgraph.c is not mentioned here in detail since the software is not central to the main objective. There are module-dependent functions that save output results to a file. A generic example of that type is discussed next.

An example of a generic function that saves the results to a file is given in Figure 3.10. The form of the function name is Save*Data where "*" is a unique name. Since this type of function is algorithm type dependent, these function are located in the individual user interface modules. The first argument to a function of this type is a pointer to a structure of type OPENFILES. This structure is declared globally and is initialized by other functions. The structure EXAMPLE contains information unique to a particular algorithm. The body of the function is a straightforward example of the use of the fprintf function in C.

```
void SaveExampleData( struct OPENFILES *pof, struct EXAMPLE *pex, ... )
{
:    Initialize any variables needed by the function.

fprintf( pof->Output, "Example Method Results\n\n");
fprintf( pof->Output, "Source: %s\n", pof->InName );
fprintf( pof->Output, "Sample Freq: %5.2fKHz Window Size: %4d\n",
     pof->SampFreq, pex->WinSize );
fprintf( pof->Output, "Start Sample: %5d Filter Order: %2d\n\n",
     pex->WinStart, pex->FilterOrder );

:    Print output results for this algorithm.

fprintf( pof->Output, "\n\n\n");

return;
}
```

Figure 3.10: Example of a Data Save Function

D. AUXILIARY FUNCTIONS

The last area that needs to be addressed in this chapter is the auxiliary software packages that do not fit into any of the other categories. There are two modules of this type. The first of these modules is Dspconv.c. This group of functions converts a DSP-16 binary data file to a DSP-16 ASCII file and *vice-versa.* The next group of functions is Dspdata.c. This group obtains the name of the input data file, the output data file, and the sampling frequency of the input file. With this file information obtained, the input data is read into an array and the output file is open for writing. These two modules use the same method for input and output described in sections B and C of this chapter. The next module to be discussed is one that contains functions used by many different modules.

The library of general use functions is contained in Dspgen.c. Functions were placed in this module if the function was short and could be used by several different modules. These functions are typically ten lines or fewer in length, are straight forward, and are self explanatory. The last module that is auxiliary to the program is Complex.c. This provides functions that perform operations on complex numbers. This group of functions is needed because complex numbers are not native to the C language as they are in FORTRAN. The last kind of function to be covered are those that are unique to a module.

Within each module there are functions that are only called by other functions in that module and are not of the type discussed in sections A, B,or C of this chapter. In this case, the functions are provided at the end of the source file for that module and are typically less than half a page long.

IV. DSP SOFTWARE

This chapter discusses the software packages that are used by the PC and the DSP-16 to perform the algorithms. These packages include the PC calculation module, the host PC to DSP-16 interface module, and the TMS320C25 assembly module. The software written for the PC side of this program use the C language—specifically Microsoft QuickC, Version 2.0. The software package for the DSP-16 uses Texas Instruments assembly language and Ariel assembly directives [Refs. 4, 7, 8].

A. PC CALCULATIONS

1. FFT

The implementation of steps given in section II.A.1 is as follows. Reference 1. pages 510-512 gives the pseudocode for the bit reverse algorithm and butterfly calculation. This pseudocode was translated into two C language functions. The functions were verified for correctness by comparing results from this program to results from Matlab. The C function CalFFTDSP contains both of these procedures. Within the function DoFFTMultiple is the control loop that offsets the input sequence in time and performs the FFT on the new input sequence.

2. Time Domain

The five algorithms described in section II.A.2 are easily implemented in C. The summation required can be achieved by a **for** loop construct. Table 4.1 cross-references the algorithm to the corresponding C function. Each of the functions were verified for correctness by comparing the results with results from Matlab.

TABLE 4.1: TIME DOMAIN FUNCTION CROSS-REFERENCE

Algorithm	Function
Short-Time Energy	CalSTEnergy
Short-Time Magnitude	CalSTMag
Short-Time Zero Crossing	CalZeroX
Short-Time Autocorrelation	CalAutoCorr
Short-Time 3-Level Autocorrelation	CalSTAutoCorr3Lvl

3. LPC

The functions to implement Durbin and Burg methods [Ref. 3, pages 554-555,564] were developed to conform with dynamic memory allocation procedures. These functions were verified by comparing the results from the functions to results given in Ref. 3, pages 209-210,261. The covariance method was implemented in C by carefully following the step-by-step procedure given in section II.A.3. The results of this function were compared against the other two functions. Table 4.2 cross-references the algorithm to the corresponding C function.

TABLE 4.2: LPC FUNCTION CROSS-REFERENCE

Algorithm	Function
Durbin Method	CalDurbin
Burg Method	CalBurg
Covariance Method	CalCovariance

B. DSP-16 AND INTERFACE SOFTWARE

The first area to be covered in this section is the programming of the DSP-16 system using assembly language programming. This is followed by a discussion of the PC software that interfaces the PC to the DSP-16.

1. DSP-16 Calculations

The first item that needs to be understood about the arithmetic on the TMS320C25 is the number format. The processor performs only integer arithmetic as a native operation. If the processor is to perform arithmetic at its maximum throughput, then integer type numbers must be used.

To meet the integer arithmetic requirement, Texas Instruments uses a number representation called the Q format. This format allows floating point numbers to be conveniently represented as integer type numbers. In this case, the integer type refers to a 16 bit representation. The most common Q format is Q15. In this format any floating point number x, where $-1 \leq x < 1$, can be represented as an equivalent 16 bit number. In the program that was developed all samples are normalized to fall in the range of $[-1, 1)$, and therefore conversion to the Q15 format is not a problem. Any number, n, can be converted to a QX format, where X a is number in the range $[0, 15]$, by multiplying n by 2^X. The number, x, must be in the range $[-2^{(16-X)}, 2^{(16-X)})$ and if $x = 2^{(16-X)}$, then it is stored as 7FFFH. Therefore in this case, the method used to convert to the Q15 format is to multiply the floating point number by 2^{15} with one exception. If the floating point number is 1, then it is stored as 7FFFh. This is the maximum positive number that can be stored as a 16-bit twos-complement number and the corresponding Q15 format number is 0.999969. To convert from Q15 to a floating point number, we divide the Q15 number by 2^{15}.

The next point about the Q15 format that must be understood concerns multiplication. When two 16-bit Q15 numbers are multiplied and stored in a 32-bit register, the result is in the Q30 format. If the upper 16 bits of the result are saved, then the saved result is a Q15 number that has been divided by two [Ref. 8, pages 76-77]. This scaling insures that an overflow will never occur, but each time this procedure is performed one bit of precision is lost. This is an important point to

note. As a consequence, each stage of a FFT automatically loses one bit of precision. For example, a 1024 FFT will only have 5 bits of precision in the end result.

The next several sections illustrate the programming techniques that can be used to program the TMS320C25 These techniques can be used in combination with each other and with other instructions to implement the desired algorithms.

a. Calculation Selection

The same basic calculation control mechanism is used by all the assembly programs. Figure 4.1 is a section of the code from the time domain calculations that illustrates the principle. The TDOMTBL is stored within the program code and is a jump table to the beginning instruction for each algorithm. During the initialization of the program, the jump table is moved to data RAM and the storage location MLOOPCONT is set to zero. The control loop is from the label MLOOP to the B MLOOP instruction (the last line in Figure 4.1). It works by first checking the memory location MLOOPCONT. If the contents are zero, then the program branches back to MLOOP and the value is checked again. If the memory contents are not zero, then we add the value at MLOOPCONT to the beginning value of the jump table. This value is the memory location in the jump table that holds the branch address of the desired algorithm. After retrieving the value at that memory location, the program branches to the algorithm. When the algorithm is complete, program execution returns to this point. After the program returns from the algorithm branch, the memory location MLOOPCONT is set to zero. The control loop now signals the PC that the algorithm is complete by setting TDR. Since the bottom of the control loop has been reached, the control loop branches to the beginning of the loop and waits for the value of MLOOPCONT to change.

```
TDOMTBL: DW    0
         DW    ENERGY
         DW    MAGIT
         DW    ZEROC
         DW    AUTOC
         DW    LEVLAC
;
;
;
; ENTRY POINT: initialization.
;
INIT:    DINT  ;redundant but safe
         :
;
         LRLK  AR7,JMPTBL      ;move jump table to data RAM
         RPTK  5               ;
         BLKP  TDOMTBL,*+      ;
;
         LRLK  AR7,MLOOPCONT;
         SACL  *               ;MLOOPCONT set to zero
;
         :
;
MLOOP:   LRLK  AR7,MLOOPCONT;if MLOOPCONT zero, then busy idle
         ZALS  *
         BLEZ  MLOOP
         ADLK  JMPTBL          ;else add value to jump table to
         SACL  DUMMY           ;determine which TDOM to call
         LAR   AR7,DUMMY
         ZALS  *               ;acc now contain to place to jump
         CALA                  ;
         LARP  AR7
         LRLK  AR7,MLOOPCONT;
         OUT   *,WRHOST        ;
         ZAC
         SACL  *
         B     MLOOP
```

Figure 4.1: Calculation Control Mechanism

b. For Loop Equivalent

In order to implement many of the desired algorithms a **for** loop mechanism is required. Figure 4.2 demonstrates one method of achieving a **for** loop control. In this method, AR0 is a countdown counter and AR1 and AR2 are used to point to operands. Each time through the loop, the counter AR0 is decremented. As long as AR0 is not zero, the program branches to NEXT41. When AR0 is zero, the program proceeds to the next instruction which branches past the arithmetic section. This figure also illustrates the multiply and accumulate process. When a branch to

```
LOOP41:   BANZ  NEXT41,*-,AR1    ;ARE MAIN LOOP NEED, YES BRANCH
          B     DONE41,*,AR0     ;NO, BRANCH TO REMAINDER
;
NEXT41:   LT    *+,AR2
          MPYA  *+
          B     LOOP41,*,AR0
;
DONE41:   APAC
```

Figure 4.2: For Loop Equivalent

the arithmetic section occurs, the operand that AR1 points to is loaded into the T register and AR1 is incremented. Next the operand that AR2 points to is multiplied by the T register. The result is stored in the P register and AR2 is incremented. The previous value of the P register is added to the accumulator. When the looping is complete, the product in the P register is added to the accumulator.

c. FFT Bit Reversal

The bit reversal algorithm is implemented as a macro given in Figure 4.3. In this macro the reverse carry mode of indirect addressing is used. This requires that AR0 be loaded with the value of half of the FFT size. In the FFT calculations, a real and an imaginary 16-bit word are stored in consecutive addresses.

Since two words of storage are required for each sample, the half size in words of storage is equal to the FFT size. The registers AR1 and AR2 are initialized to point to the real value of the first data element. AR3 is used as a countdown counter and AR4 points to a storage location. The first step is to check if the pointer in AR1 is greater than the pointer in AR2. If this is the case, a branch to NOMOV is made. At NOMOV the AR1 pointer is incremented twice. This points it to the next real value. Between the BLEZ NOMOV= and BACK=, the exchange of values takes place using the accumulator as temporary storage. At BACK= the FFT size is added to AR2 with reverse carry. The next step tests to see if the loop is complete. If so, then it branches to FINISH; otherwise it branches to LOOP=.

2. Assembly Debugging

After an algorithm was written and assembled successfully, the next stage was to debug the algorithm on the board. This was accomplished by inserting the program to be tested into RESMON.ASM which stands for resident monitor. Note that RESMON.ASM is a program that Ariel provides with the DSP-16 system. The commands used by this program are in [Ref. 4, Chapter 9]. In order to use RESMON.ASM the origin of the test program must be placed after location 1024. Therefore any variables that would normally be in memory locations lower than 1024 must be relocated for testing. We note that the test program must be physically inserted into the RESMON.ASM because the assembly is non-relocatable and does not use a linker. This means that, even for short test programs, one must always reassemble the 14,920 instruction words required by the monitor.

At this point, the PC side of the monitor can be started. The PC program is called DSPBUG.EXE. Once this program initializes the board and loads the monitor with the test program onto the board, the monitor commands and utilities can be used to check the program execution. At this point in the verification, the best procedure is

```
BITRVI:   MACRO                 ;FIRST,BIAS,FFTSIZE,TEMPBUF
;
;         INITIALIZE AUXILIARY REGISTERS
;
          LRLK  AR0,$2          ;AR0 IS LENGTH OF FFT
          LRLK  AR1,$0+$1       ;AR1 POINTS TO FIRST PT
          LRLK  AR2,$0+$1       ;AR2 POINTS TO FIRST PT
          LRLK  AR3,$2-1        ;LOOP CONTROL FFTSIZE - 1
          LRLK  AR4,$3          ;TEMP BUFFER PTR
          LARP  AR4
;
LOOP=:    SAR   AR1,*
          ZALS  *
          SAR   AR2,*
          SUBS  *,AR1
          BLEZ  NOMOV=
          ZALH  *,AR2
          ADDS  *,AR1
          SACL  *+,0,AR2
          SACH  *+,0,AR1
          ZALH  *,AR2
          ADDS  *,AR1
          SACL  *+,0,AR2
          SACH  *-,0
BACK=:    MAR   *BR0+,AR3       ;THIS IS NEEDED TO *BR0+ ON AR2
          BANZ  LOOP=,*-,AR4
          B     FINISH=
NOMOV=:   MAR   *+
          MAR   *+,AR2
          B     BACK=
;
FINISH=:
```

Figure 4.3: FFT Bit Reversal

to manually enter data points into the board's data RAM at the appropriate location. The test data values should be small in number and provide results that are easy to follow. Then the test program can be single stepped and the registers can be observed. This procedure will identify incorrect logic and instruction usage. Additionally, the numerical results at each stage can be followed. As errors are found, the corrections are made to the program under test and the above procedure repeated.

After the program is tested using the above procedure, the test program can be be separated into a stand-alone program. The stand-alone must be assembled as a stand-alone program. Now a larger and more realistic set of data values can be loaded using the PC interface functions explained in the following section. The results can be returned to the PC and compared against known results. The known results used in all cases were provided by Matlab. The FFT results (partial listing) given in Table 4.3 shows the anticipated loss of precision as the number of stages increased.

TABLE 4.3: FFT RESULTS COMPARISON

FFT Size	Result	Matlab	DSP-16
2	$x_{out}(0)$	$-0.0383 + j0.0000$	$-0.0383 + j0.0000$
	$x_{out}(1)$	$0.0112 + j0.0000$	$0.0112 + j0.0000$
8	$x_{out}(0)$	$-0.2483 + j0.0000$	$-0.2483 + j0.0000$
	$x_{out}(1)$	$0.0412 - j0.0154$	$0.0410 - j0.0156$
32	$x_{out}(0)$	$-0.3409 + j0.0000$	$-0.3418 + j0.0000$
	$x_{out}(1)$	$-0.1413 + j0.1846$	$-0.1445 + j0.1826$
128	$x_{out}(0)$	$-2.3580 + j0.0000$	$-2.3633 + j0.0000$
	$x_{out}(1)$	$-0.4993 + j0.2542$	$-0.5195 + j0.2383$
512	$x_{out}(0)$	$-8.2644 + j0.0000$	$-8.2969 + j0.0000$
	$x_{out}(1)$	$-1.1061 + j0.7113$	$-1.2031 + j0.6563$
1024	$x_{out}(0)$	$-18.4322 + j0.0000$	$-18.5000 + j0.0000$
	$x_{out}(1)$	$-0.8486 - j1.1209$	$-1.0938 - j1.2813$

3. Host Interface Software

One of the software packages that Ariel provides with the board is a set of interface functions. A description of the functions and the arguments sent and returned by the functions are given in [Ref. 4, Chapter 4]. The software version used by this program is contained in DSP11MCL.LIB. In order to simplify the function calls and structure them like a C function, a library of C interface functions was developed. Figure 4.4 gives an example of one of the functions. The function arguments are variables that change from function call to function call. Within the function these arguments are stored in the appropriate structure member and then a call is made to

```
BOOL ExecDSP( int NumArg, int *argument )
{
BOOL ret = FALSE;
char buffer[39];
struct parms dcall;

dcall.fun = DDARR ;
dcall.p2 = 0x3ffc ;
dcall.p3 = NumArg ;
dcall.p6 = argument ;

if( ! dsp16(&dcall) )
     {
     sprintf( buffer, "DSP Error: Function %d, Return %d", dcall.fun,
          dcall.retval );
     ErrorMsg( buffer);
     ret = TRUE ;
     }

return ret;
}
```

Figure 4.4: C Interface Function

the dsp16 function contained in the Ariel library. If the dsp16 function return value is zero, then an error occurred. In this case an error message is processed by the ErrorMsg function and the variable "ret" is set to TRUE. In any case, the current value of "ret" is returned to the calling function. A return value of TRUE means that an error occurred in the function call while FALSE means no error. By using these interface functions the steps needed to interface to the board can be easily achieved.

In order to use the same user-interface developed in Chapter III the function calls to the DSP-16 version of the calculation functions must be the same as the PC version developed in the section above. This can be accomplished by using the same function name and argument list. The internal coding of the two functions bear very little resemblance to one another. In the PC version the actual calculations are performed while in the DSP-16 version only the setup is designed so that the board can perform the calculations. An actual example of this setup is given in Figure 4.5. The first stage of the setup is to ensure that the proper software is loaded onto the board. In this case the software is in the file TDOM.HEX. The next step is to allocate memory for the Q15 array and then convert the floating point array to a Q15 array. If an error condition is noted during any function call to the board, then the program will exit with an exit code of one. The next several steps serve to download the data, pass to the board an indication of which algorithm to run and any additional arguments needed for that algorithm, wait for the board to complete the calculation, and upload the results. The results are then converted back to the floating point representation. In this example the result was normalized and the number of shifts needed to normalize it was also stored before the result was removed from the accumulator of the TMS320C25. Both the normalized result and the shifts are used in the format conversion formula to reduce the loss of precision.

```
float CalSTEnergy( int WindowSize, float *X )
{
float temp = 0.0 ;
QFORMAT *qinput;
int i,arguments[] = { 1, 0 };
static char *pszLoadFile = { "TDOM.HEX" };
arguments[1] = WindowSize;
if( iTypeDSP != TIMEDOMAIN )
    {
    if( LoadDSPWithReset( pszLoadFile ) )
        {
        exit(TRUE);
        }
    iTypeDSP = TIMEDOMAIN;
    }
qinput = (QFORMAT *)GetMem( WindowSize, sizeof(QFORMAT) );
for( i = 0 ; i < WindowSize ; i++)
    {
    qinput[i] = ftoq14(*X++);
    }
if( PCtoDSP( WindowSize, qinput, TDOMLOADDATA ) )
    {
    exit(TRUE);
    }
if( ExecDSP( 2, arguments ) )
    {
    exit(TRUE);
    }
if( WaitForDSP( ) )
    {
    exit(TRUE);
    }
if( DSPtoPC( 2, qinput, TDOMLOADDATA ) )
    {
    exit(TRUE);
    }
temp = qtof(qinput[0], ( 0x0001 << (6+qinput[1]) ));
free(qinput);
return( temp );
}
```

Figure 4.5: DSP-16 Version of Calculation Function

4. Interface Debugging

The debugging at this point has an added difficulty not present in most programming environments. When an error occurs, is the error in the PC software or in the DSP-16 software? One procedure that can help to reduce this uncertainty is to write a dummy function to replace the library function DSP16. By using this dummy function the PC software can be tested down to the board calls. If the procedure outlined in the Assembly Debugging section is used to test the assembly programs, then the algorithm can be tested. If an error occurs in this situation, the most likely cause of the error is memory location assignments. Recall from the discussion of the monitor (section B.2 of this chapter) it was mentioned that the actual memory location used by various variables might have to change for testing procedures. At this point the only solution is to carefully check all the memory location assignments. This has been simplified by defining the locations in an "include file" for the PC and a macro file for the board.

This chapter has presented the procedures for developing and interfacing the calculation software for both the PC version and the DSP-16 version. With these software packages and the user interface software packages, the complete program can be assembled. If the DSP-16 version is linked in at assembly time, then the program will be a DSP-16 version, otherwise the normal (PC) version will be linked and the result will be a be a PC-only version of the program.

V. CONCLUSIONS

The objectives of this thesis, namely to develop a DSP program that requires no programming by the user and to be able to utilize the Ariel DSP-16 system as a pseudo coprocessor, have been met. The common user-interface goal has been achieved. In addition, the user interface has been found by the students to be quite user-friendly. The final interface evolved over time based on feedback received from an EC4410 class that used early versions of the program. The ultimate proof of the common user-friendly-interface can best be shown by actually running both versions of the program. The algorithm package for the PC implements FFT algorithms of various sizes, five different time domain calculations, and three different linear predictive coding methods. The package for the DSP-16 has a complete interface package and demonstrates the basic building blocks for all the algorithms. The package for the DSP-16 has all the FFTs and time domain calculations implemented. Feedback from the EC4410 class was very helpful in detecting bugs. The complete software package is available on request from Department of Electrical and Computer Engineering, Code 62Tu, Naval Postgraduate School, Monterey, California 93943-5002. The following sections present a discussion of observations made during this research and possible further developments to improve on the work reported here.

A. OBSERVATIONS

1. TMS320C25 Limitations

The size of onboard processor RAM determines the largest window size that can be used. Window sizes larger than the amount that can be accommodated by the onboard processor RAM require internal to external data RAM swapping or

total use of external data RAM. Either of these options greatly slows the throughput. For this processor the maximum window size which does not require any of these two options is 512 real words or 256 complex words. The remaining 32 words are reserved for overhead.

Another difficulty is the loss of precision during stages of multiplications. If, for an FFT, the window size is chosen as 256 due to RAM considerations, then the result automatically has a loss of eight bits of precision. There are ways to overcome this problem. One method is to normalize the intermediate results after every stage of multiplication. This method requires twice the amount of RAM and the normalization reduces throughput considerably. Another method is to use an IEEE floating point representation for the calculations. For the single precision IEEE floating point representation, the storage required for each sample is four words [Ref. 8, pages 245-268]. Note that this is twice the normal storage for a single precision number. For this method, therefore, four times the initial RAM is required and the throughput will again be reduced. The alternative to more RAM would be a corresponding window size reduction.

2. DSP-16 Limitations

As previously mentioned, the DSP-16 has only 16K words of external RAM that can be used for program and/or data storage. For our implementation, the limitation is only for FFT calculations, but nevertheless is a significant limitation. Initially, the 256-, 512-, and 1024-point FFTs were to use a 256-point FFT subroutine as the basic building block. After assembling the subroutine, it became clear that it was not possible as the subroutine required 21,776 words of storage. Next a 128-point FFT subroutine was tried. The assembled output required 9194 words. However, when the code to perform the 128, 256, 512, and 1024 FFTs was added, the total size was 12680 words. Since the control overhead is only one word, it is not possible to

fit this 12680 word program together with 1024 complex words of data and control overhead in the available RAM. In order to fit the 12680 word program in program RAM, the GREG register must be set to F8H. With this GREG setting and one control word, there is only 2047 words of RAM left for data which is one word short of the required 2048 words of data RAM needed. The final solution was a 64 point FFT subroutine. This requires considerable amounts of data transfers and therefore reduces the system throughput. Since the TMS320C25 can address upto 64K words of program memory and 64K words of data memory, the limitation of 16K words for both program and data RAM imposed by the board is not advisable.

3. Assembler and Linker

Since the assembler provided by Ariel is a non-relocatable type and therefore does not require a linker, a source code module had to contain all the required instructions. This required continual reassembly of large sections of code that were already debugged. The assembler sold by Texas Instruments for this processor (and widely used) is a relocatable type with a linker. This seems like a more reasonable approach to program development.

B. FURTHER DEVELOPMENTS

There are several improvements that are desirable and areas that need further work. The first improvement is for the user's benefit. In the present form, the program will give an error message, if the entered value is not in an acceptable range for integer and real data entry prompt. The entry functions can be changed so that the acceptable values are displayed at the bottom of the screen when the entry prompt is displayed. The error message can still be used to alert the user to an invalid entry. The next improvement concerns the output display for multiple window FFTs. Instead of displaying one window at a time, based on user entry, all

windows could be displayed at one time in a 3-D plot where the third axis is time. Although the programming of a 3-D plotting function is no simple matter, when a plotting function of this type is available, the present display function can easily be changed to incorporate the new plotting function.

The last two changes deal with the DSP-16 version only. In the DSP-16 version of the program, the non-real-time LPC algorithms are still performed by the PC. These algorithms need to have assembly versions developed and then interfaced with the PC as has been done for the FFT and time domain calculations.

The last change is to step from non-real time calculations to (near) real-time calculations. The sections of assembly code that actually perform the calculations should need no modification, but the control loop would need to be totally changed. As already has been pointed out however, the ability of this system to perform real-time FFTs of sizes above 64 points is questionable because of inherent hardware limitations.

REFERENCES

1. Strum, Robert D. and Kirk, Donald K., *First Principles of Discrete Systems and Digital Signal Processing*, Addison Wesley Publishing Co., Reading, MA, 1988.

2. Rabiner, Lawrence R. and Schafer, Ronald W., *Digital Processing of Speech Signals*, Prentine-Hall, Inc., Englewood Cliffs, NJ, 1978.

3. Orfanidis, Sophocles J., *Optimum Signal Processing: An Introduction*, MacMillan Publishing Co., New York, NY, 1988.

4. Ariel Corp., *Operating Manual for the DSP-16 Data Acquisition Processor*, Ariel Corp., 1987.

5. Microsoft Corp., *Microsoft QuickC: C for Yourself, Version 2.0*, Microsoft Corp., 1988.

6. Microsoft Corp., *Microsoft QuickC: Tool Kit, Version 2.0*, Microsoft Corp., 1988.

7. Texas Instruments Inc., *TMS320C25 User's Guide, Preliminary*, Texas Instruments Inc., 1986.

8. Texas Instruments Inc., *Digital Signal Processing Applications with the TMS320 Family, Theory, Algorithm, and Implementations*, Texas Instruments Inc., 1986.

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5002	1
4.	Professor Murali Tummala, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5002	3
5.	Professor C. W. Therrien, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5002	1
6.	Professor Ralph Hippenstiel, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5002	1
7.	Mr. Les Listwa Ariel Corporation 433 River Road Highland Park, New Jersey 08904	1

8. Mr. Panos Papamichalis — 1
Texas Instruments, Inc.
Box 1443, Mailstop 701
Houston, Texas 77451-1443

9. MAJ Thomas DeMars — 1
PM C2G MCRDAC
MCCDC
Quantico, Virginia 22134

10. LCDR Gregory J. Pitman — 2
Carrier Airborne Early Warning Squadron 110
NAS Miramar
San Diego, California 92145